# Programación Python para Machine Learning

## Lección 11: Ajuste de hiperparámetros y flujos de trabajo en Machine Learning.

eip ESCUELA INTERNACIONAL POSGRADOS

# ÍNDICE

# Lección 11: Ajuste de hiperparámetros y flujos de trabajo en Machine Learning.

## 1. Introducción

Un proceso completo de Machine Learning consiste, además de entrenar un modelo, en seleccionar aquellos parámetros que hacen que tenga un rendimiento óptimo sobre un conjunto de datos en particular. Esta evaluación del rendimiento de los predictores no siempre es una tarea directa ya que hay varios factores que determinan el rendimiento de los modelos.

Por otro lado, los flujos de trabajo constituyen la línea argumental de todo proceso de Machine Learning. Diseñarlos, estructurarlos y configurarlos adecuadamente, incidirá positivamente en muchos aspectos de propio proceso.

En esta lección se explicará los motivos por los que se hace necesario un correcto ajuste de hiperparámetros y se explorará en detalle distintas estrategias para seleccionar automáticamente los mejores parámetros para un modelo. Además, se explicará la conveniencia de diseñar un buen flujo de trabajo y cómo implementarlos adecuadamente en Python.

## Objetivos

- Conocer los motivos fundamentales por los que es necesario llevar a cabo un proceso de ajuste de parámetros en modelos supervisados de Machine Learning.
- Dominar las técnicas de implementación de los métodos más comunes de ajuste de hiperparámetros en Python.
- Describir qué es un flujo de trabajo en Machine Learning y su utilidad.
- Saber utilizar las técnicas de implementación de flujos de trabajo en Machine Learning en Python.

## 2. AJUSTE DE HIPERPARÁMETROS

El rendimiento de la mayoría de los modelos supervisados de Machine Learning dependen de una serie de hiperparámetros que controlan su manera de comportarse. El valor de estos hiperparámetros ha de ser ajustado previamente a proceder con la fase de entrenamiento. Este ajuste definirá de manera directa el rendimiento del modelo sobre un conjunto de datos determinado.

Al contrario de lo que ocurre con los parámetros del modelo, aprendidos y concretados durante la fase de entrenamiento, existe una dificultad intrínseca en la tarea de encontrar de los valores asociados a los hiperparámetros. Esto se debe a que los hiperparámetros óptimos de cada modelo dependen de la naturaleza del conjunto de datos que conforman el problema de predicción, no habiendo un procedimiento universal con el que ser establecidos.

Además, la correcta comparativa del rendimiento de diferentes modelos predictivos sobre un conjunto de datos implica llevar a cabo un ajuste adecuado de hiperparámetros con el objetivo de que el análisis se realice en el contexto en el que todos los modelos están proporcionando lo mejor de sí.

A todo ello se une que, salvo en algunos casos, los hiperparámetros de los modelos son más de uno y el efecto que producen no lo suele ser de manera independiente al resto. Esto se traduce en que, en la práctica, lo que se debe establecer es un conjunto de hiperparámetros y no cada uno de manera autónoma.

De manera bastante frecuente, se omite el ajuste de hiperparámetros, permaneciendo estos con el valor establecido por defecto en la biblioteca con la que se implementa el método. Otra opción que se suele dar es la de buscar aleatoriamente mediante prueba y error el valor óptimo de hiperparámetros. Sin embargo, este proceso es muy exigente y no garantiza encontrar unos valores adecuados.

De todo esto se deduce que, en lugar de seleccionar por defecto o aleatoriamente los valores de los hiperparámetros, un mejor enfoque sería considerar la tarea como un problema de búsqueda y establecer un procedimiento que trate de encontrar de modo sistemático la combinación óptima de hiperparámetros para un modelo determinado sobre un conjunto de datos en particular. Este proceso se denomina optimización de hiperparámetros, ajuste de hiperparámetros o búsqueda de hiperparámetros.

Todo proceso de búsqueda implica definir una representación de las soluciones, un espacio donde realizarla y un objetivo. En este caso, cada solución representará una configuración del modelo.

El espacio de búsqueda viene definido geométricamente como un volumen de n-dimensional, siendo n el número de hiperparámetros a considerar y la escala de la dimensión los valores que puede tomar cada uno de los hiperparámetros.

El objetivo, encontrar un vector de valores de hiperparámetros que maximice el rendimiento del modelo tras el proceso de aprendizaje.

Aunque a veces se utilizan métodos más avanzados, como la computación evolutiva, normalmente se utilizan dos de los métodos más sencillos y populares:

1. **Búsqueda en rejilla (grid).**

2. **Búsqueda aleatoria**

**Búsqueda en rejilla (grid):** Se define un espacio de búsqueda como una cuadrícula de valores de hiperparámetros y se evalúa cada posición de la cuadrícula (véase Figura 2.1). Finalmente, se elige la combinación de hiperparámetros con la que el modelo ha ofrecido mejor rendimiento.

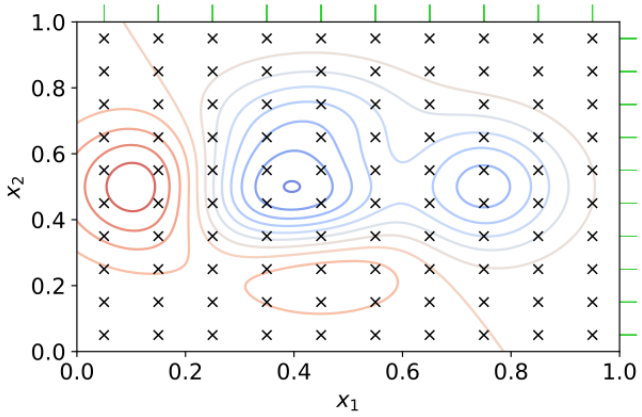


*Figura 2.1: Ajuste de dos hiperparámetros mediante Búsqueda en rejilla.*

**Búsqueda aleatoria.** Se define un espacio de búsqueda como un dominio acotado de valores de hiperparámetros y se toman muestras aleatorias de puntos continuos dentro de ese dominio (véase Figura 2.2). De igual modo, se elige la combinación de hiperparámetros con la que el modelo ha ofrecido mejor rendimiento.

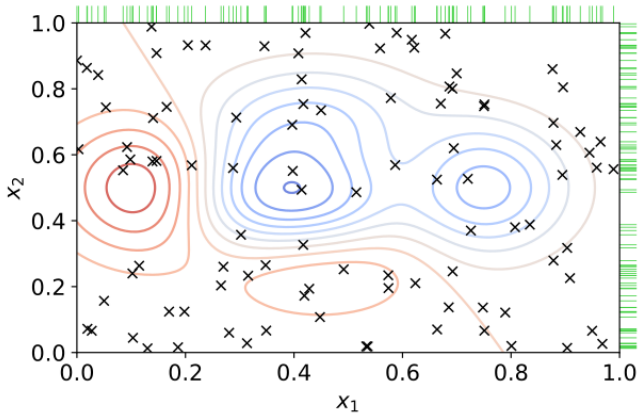


*Figura 2.2: Ajuste de dos hiperparámetros mediante Búsqueda aleatoria.*

# 3. Implementación en Python de un proceso de ajuste de hiperparámetros

La librería scikit-learn proporciona técnicas para ajustar los hiperparámetros de un modelo mediante búsqueda grid o búsqueda aleatoria. Concretamente, dentro del módulo `model_selection` contiene las clases `GridSearchCV` para la búsqueda *grid* y `RandomizedSearchCV` para la búsqueda aleatoria.

## 3.1. Implementación de un proceso de ajuste de hiperparámetros mediante búsqueda en grid.

Dentro del módulo `model_selection` de scikit-learn se puede encontrar la clase `GridSearchCV` para implementar un ajuste de hiperparámetros mediante una búsqueda *grid.* Esta clase requiere de dos argumentos. Por una parte, el modelo que se trata de optimizar, y por otro, el espacio de búsqueda en forma de diccionario Python, en el que las claves son los hiperparámetros y los valores una lista de posibles valores.

Otros dos parámetros que resultan de interés a definir son el parámetro `cv`, que permite especificar un objeto de validación cruzada configurado, y el parámetro `scoring` que indica la métrica a optimizar.

```
from pandas import read_csv
from sklearn.svm import SVC
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV
from random import seed
from time import time

seed= seed(time())
filename = '../datasets/seeds.data'

data = read_csv(filename, sep="\t")
```

```
X = data[data.columns[:-1]]
Y = data[data.columns[-1]]

model = SVC()
cv=RepeatedStratifiedKFold(n_splits=10, n_repeats=3 )

space =  [{'kernel':['poly', 'sigmoid', 'rbf'],
           'C' :[1e-3, 0.1, 1, 1e1, 1e2, 1e4],
           'class_weight': [None,'balanced'],
           'gamma': ['auto', 'scale']},
          {'kernel':['linear'],
                    'C' :[1e-3, 0.1, 1, 1e1, 1e2, 1e4],
                    'class_weight': [None,'balanced'],
                    }
          ]

search = GridSearchCV(model,
                      param_grid=space,
                      scoring='balanced_accuracy',
                      n_jobs=-1,
                      cv=cv)
result = search.fit(X, Y)
```

## 3.2. Implementación de un proceso de ajuste de hiperparámetros mediante búsqueda en aleatoria.

Dentro del módulo `model_selection` de scikit-learn se puede encontrar la clase `RandomizedSearchCV` para implementar un ajuste de hiperparámetros mediante una búsqueda aleatoria. Esta clase requiere de dos argumentos. Por una parte, el modelo que se trata de optimizar, y por otro, el espacio de búsqueda en forma de diccionario Python, en el que las claves son los hiperparámetros y los valores una lista o una distribución de valores a muestrear.

Otros tres parámetros que resultan de interés a definir son `cv`, que permite especificar un objeto de validación cruzada configurado, el parámetro `scoring` que indica la métrica a optimizar, y el número total de puntos a valorar en el espacio de búsqueda, mediante el parámetro `n_iter`.

```
from pandas import read_csv
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import RandomizedSearchCV
from random import seed
from time import time
from scipy.stats import loguniform, randint

seed= seed(time())
filename = '../datasets/seeds.data'
data = read_csv(filename, sep="\t")

X = data[data.columns[:-1]]
Y = data[data.columns[-1]]

model = DecisionTreeClassifier()
cv=RepeatedStratifiedKFold(n_splits=10, n_repeats=3 )

searchSpace =  [{'criterion':['entropy'],
           'max_depth' :randint(3,30),
           'class_weight': [None,'balanced'],
           'ccp_alpha':  loguniform(1e-5, 100) }
          ]
search = RandomizedSearchCV(model,

param_distributions=searchSpace,
                            scoring='balanced_accuracy',
                            n_jobs=-1,
                            cv=cv,
                            n_iter=500)
result = search.fit(X, Y)
```

## 3.3. Consideraciones prácticas sobre el proceso de ajuste de hiperparámetros.

A la hora de implementar un proceso de ajuste de hiperparámetros, bien mediante búsqueda *grid*, bien mediante búsqueda aleatoria, es aconsejable definir y especificar un objeto de validación cruzada para asumir más control sobre la evaluación del modelo y hacer que el procedimiento de evaluación sea obvio y explícito. En el caso de las tareas de clasificación, se recomienda utilizar la clase `RepeatedStratifiedKFold`, y para las tareas de regresión utilizar `RepeatedKFold` con un número adecuado de folds y repeticiones.

# 4. Flujos de trabajo en Machine Learning

Un proyecto completo de Machine Learning normalmente incluye un proceso previo de depuración del conjunto de datos a distintos niveles. Una vez el conjunto de datos está listo, completo y estructurado, se puede proceder con algún tipo de preprocesamiento, tales como la transformación del tipo de atributo, selección de características, escalado o normalización de variables, incluso con reducción de dimensionalidad.

En algún caso, puede ser que tal preprocesamiento solo necesite de una de estas acciones. Sin embargo, lo habitual suele ser que se necesite enlazar varias de estas etapas juntas, para finalmente proceder con el entrenamiento, ajuste y evaluación de un modelo de predicción.

El objetivo es proceder con una secuencia lineal de transformaciones de datos que termina en un proceso de generación de un modelo predictivo que sea evaluado garantizando que todos los pasos de la secuencia se limiten a utilizar los datos adecuados para una correcta evaluación del rendimiento.

Los flujos de trabajo son un instrumento de diseño simple pero generalmente muy práctico para articular todo el proceso. Dentro de un marco de trabajo en Machine Learning, llevar a cabo un flujo de trabajo adecuado redunda en varios aspectos:

- Es adecuado para disponer de una línea de ejecución robusta y conceptualmente fácil de entender.
- Facilita seguir el orden adecuado en la aplicación de los distintos pasos.
- Proporciona la capacidad de que el proceso pueda ser replicado.

## 4.1. Implementación en Python de un flujo de trabajo

La clase `Pipeline` del módulo `pipeline` de Scikit-learn está implementada para completar un flujo de trabajo completo de Machine Learning en la que se incluya aplicar una serie de transformaciones de datos seguidas por el modelado de un predictor.

```
from pandas import read_csv
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV
from random import seed
from time import time

seed= seed(time())
filename = '../datasets/seeds.data'
data = read_csv(filename, sep="\t")

X = data[data.columns[:-1]]
Y = data[data.columns[-1]]

pipe = Pipeline([('scaler',  StandardScaler()),
             ('MLPClassifier', MLPClassifier(random_state =
seed))])

space = [{
    'MLPClassifier__solver': ['lbfgs', 'adam'],
    'MLPClassifier__max_iter': [2000,3000,5000],
    'MLPClassifier__activation' :
['relu','logistic','tanh'],
    'MLPClassifier__hidden_layer_sizes':[(10,), (100,)],
}]

cv = RepeatedStratifiedKFold(n_splits=5, n_repeats=3,
random_state=seed)

grid = GridSearchCV (estimator = pipe,
                     param_grid = space,
                     cv = cv,
                     return_train_score=True,
                     n_jobs=-1)

result = grid.fit(X, Y)
```

## 5. PUNTOS CLAVE

En esta lección se ha aprendido:

- Conocer los motivos fundamentales por los que es necesario llevar a cabo un proceso de ajuste de parámetros en modelos supervisados de Machine Learning.
- Dominar las técnicas de implementación de los métodos más comunes de ajuste de hiperparámetros en Python.
- Describir qué es un flujo de trabajo en Machine Learning y su utilidad.
- Saber utilizar las técnicas de implementación de flujos de trabajo en Machine Learning en Python.